

Primeiro vamos criar o atributo Job, que é um objeto que sempre vai retornar quando usar uma coroutina

class MainActivity : AppCompatActivity() {  
  
 private val binding by *lazy* **{** ActivityMainBinding.inflate(*layoutInflater*) **}** private var job: Job? = null

Agora fizemos um botão para cancelar a coroutina

binding.btnCancelar.setOnClickListener **{** job?.cancel()  
**}**

e aqui esta nossa coroutina

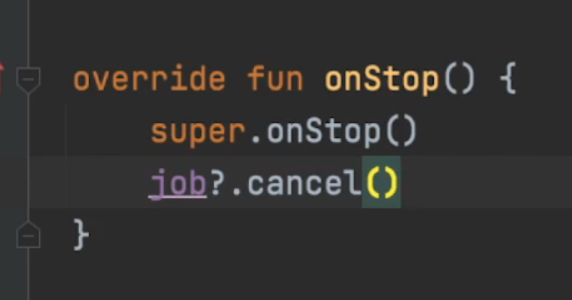
binding.btnIniciar.setOnClickListener **{** */\*\* Parar uma coroutine  
 \* Toda vez que utlizamos uma coroutina ela retorna um objeto do tipo Job  
 \* E com esse objeto do tipo Job podemos fazer algumas coisas.  
 \*  
 \*/* job = *CoroutineScope*(Dispatchers.IO).*launch* **{** executar()  
  
 **}**  
 **}**

Atenção esse método também serve para quando a gente quiser parar uma execução do tipo?

Imagina que estamos mudando de tela e temos o método onStop, e ai podemos entrar com o job para parar uma coroutina

No onStart podemos iniciar nossa coroutine e no onStop podemos parar ela.

Exemplos a baixo



Agora para iniciar uma coroutine podemos usar o método onStart que é o mais indicado e podemos fazer dessa forma que esta ai em cima, mas carregando os dados.

Reparar aqui

# Pratica para pausar coroutine com o onStop



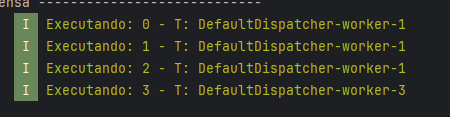
Quando a gente clicar em abrir outra tela, vamos parar a coroutine

Quando clicar em voltar queremos iniciar ela novamente

binding.btnAbrirOutraTela.setOnClickListener **{** val intent = Intent(this, SegundaActivity::class.*java*)  
 startActivity(intent)  
**}**

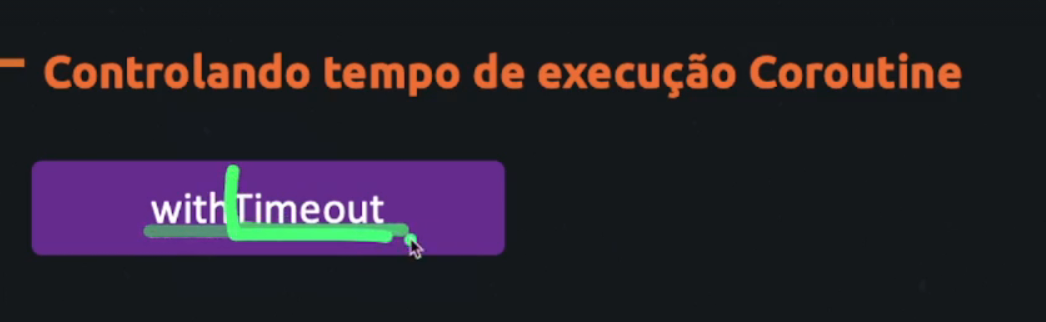
*/\*\* OnStop parando uma coroutine quando a gente abrir outra activity \*/*override fun onStop() {  
 super.onStop()  
 job?.cancel()  
}

Observa que ela fica travado, quando vamos para outra activity

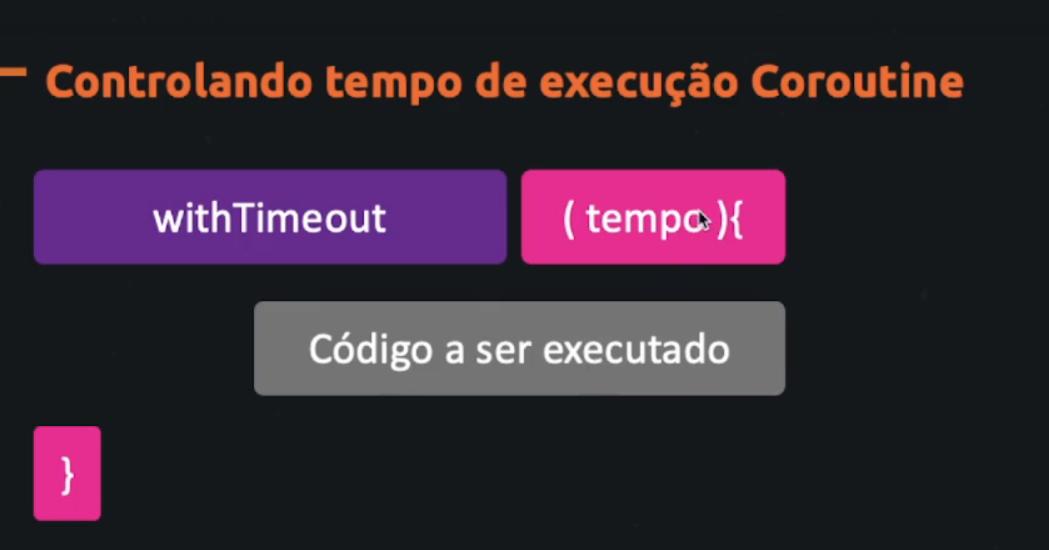


# Controlar o tempo de execução de uma coroutine

Podemos definir um tempo máximo que ela execute e quando esse tempo máximo acabar podemos encerrar a coroutine



Pra isso vamos usar esse método que quer dizer: Com limite de tempo

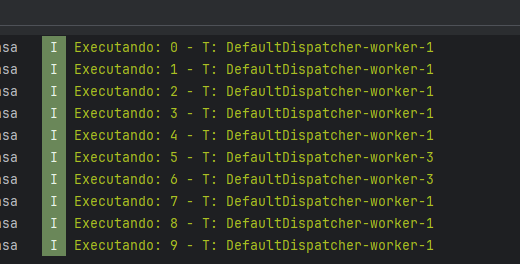


Agora vamos fazer um teste, vamos colocar para repetir 15x o código com o delay de 2 segundos, isso daria o total de 30 segundos, mas vamos colocar um tempo máximo para a coroutine trabalhar que é de 20 segundos, assim o máximo de que iria contar seria de 0 a 10.

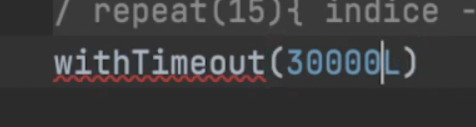
## PRATICA LIMITAR O TEMPO DA COROUTINE

*/\*\* Contando a coroutine, fazendo ela trabalhar no maximo 20 segundos \*/*job = *CoroutineScope*(Dispatchers.IO).*launch* **{** withTimeout(20000)**{** // 20 segundos.  
 executar()  
 **}  
  
}**

Repara que configuramos o executar para repetir de 2 em 2 segundos ate 15 vezes, mas expirou o tempo de 20 segundos, então ele conseguiu repetir apenas 9 vezes



Atenção:



Devemos por o L no final para indicar que é do tipo LONG e sem o L ele vai entender que é do tipo INTEIRO.